
/ This is \
\
\ ducksay! /

\
 \
 >(')
)/
 /(
 / '----/
 \ ~== /
 ~~~~~

-----  
( But which Version? )  
-----

\  
 >()\_  
 (\_\_)\_\_

-----  
( v2.2 )  
-----

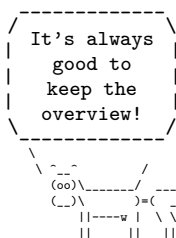
^\_\_^  
 /oo) /  
 /(\_\_)  
 / \ (   
 | w----| |  
 | | | |

-----  
( by Jonathan P. Spratte )  
-----

/  
 .-----,  
 .'\_/\_|\_\ ' ,  
 /<::[0]8::>>\  
 |-----|  
 | | -==----- | |  
 | | =====-- | |  
 \ ||( )|::: | /  
 | ||( )|... | |  
 | |-----| |  
 | | \\_\_\_\_\_/ | |  
 / \ / \ / \ \  
 (\_\_\_\_) (\_\_\_\_) (\_\_\_\_)

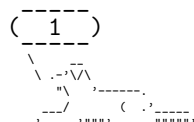
-----  
( Today is 2018-12-29 )  
-----

\ .\|//|\\| |.  
 \ |/\|/|/|/|/|  
 /. '|\|/|/|/|/|  
 o\_\_.\_|//|/|\\| |'



# Contents

|          |                                           |           |
|----------|-------------------------------------------|-----------|
| <b>1</b> | <b>Documentation</b>                      | <b>2</b>  |
| 1.1      | Downward Compatibility Issues             | 2         |
| 1.2      | Shared between versions                   | 2         |
| 1.2.1    | Macros                                    | 2         |
| 1.2.2    | Options                                   | 3         |
| 1.3      | Version 1                                 | 5         |
| 1.3.1    | Introduction                              | 5         |
| 1.3.2    | Macros                                    | 5         |
| 1.3.3    | Options                                   | 5         |
| 1.3.4    | Defects                                   | 6         |
| 1.4      | Version 2                                 | 7         |
| 1.4.1    | Introduction                              | 7         |
| 1.4.2    | Macros                                    | 7         |
| 1.4.3    | Options                                   | 7         |
| 1.5      | Dependencies                              | 12        |
| 1.6      | Available Animals                         | 12        |
| 1.7      | Miscellaneous                             | 13        |
| <b>2</b> | <b>Implementation</b>                     | <b>14</b> |
| 2.1      | Shared between versions                   | 14        |
| 2.1.1    | Variables                                 | 14        |
| 2.1.1.1  | Integers                                  | 14        |
| 2.1.1.2  | Sequences                                 | 14        |
| 2.1.1.3  | Token lists                               | 14        |
| 2.1.1.4  | Boolean                                   | 14        |
| 2.1.1.5  | Boxes                                     | 14        |
| 2.1.2    | Regular Expressions                       | 14        |
| 2.1.3    | Messages                                  | 15        |
| 2.1.4    | Key-value setup                           | 15        |
| 2.1.5    | Functions                                 | 15        |
| 2.1.5.1  | Generating Variants of External Functions | 15        |
| 2.1.5.2  | Internal                                  | 16        |
| 2.1.5.3  | Document level                            | 17        |
| 2.1.6    | Load the Correct Version and the Animals  | 18        |
| 2.2      | Version 1                                 | 19        |
| 2.2.1    | Functions                                 | 19        |
| 2.2.1.1  | Internal                                  | 19        |
| 2.2.1.2  | Document level                            | 21        |
| 2.3      | Version 2                                 | 22        |
| 2.3.1    | Messages                                  | 22        |
| 2.3.2    | Variables                                 | 22        |
| 2.3.2.1  | Token Lists                               | 22        |
| 2.3.2.2  | Boxes                                     | 22        |
| 2.3.2.3  | Bools                                     | 22        |
| 2.3.2.4  | Coffins                                   | 22        |
| 2.3.2.5  | Dimensions                                | 22        |
| 2.3.3    | Options                                   | 23        |
| 2.3.4    | Functions                                 | 24        |





```
\AddAnimal
```

adds `<animal>` to the known animals. `<ascii-art>` is multi-line verbatim and therefore should be delimited either by matching braces or by anything that works for `\verb`. If the star is given `<animal>` is the new default. One space is added to the begin of `<animal>` (compensating the opening symbol). For example, `snowman` is added with:

```
\AddAnimal{snowman}
{ \
  \ _[_]_
    (")
  >-( : )-<
    ( _ : _ ) }
```

It is not checked whether the animal already exists, you could therefore redefine existing animals with this macro.

The symbols signaling the speech (in the `snowman` example above the two backslashes) should at most be used in the first three lines, as they get replaced by `0` and `o` for `\duckthink`. They also shouldn't be preceded by anything other than a space in that line.

\AddColoredAnimal

It does the same as `\AddAnimal` but allows three different colouring syntaxes. You can use `\textcolor` in the `{ascii-art}` with the syntax `\textcolor{<color>}{<text>}`. Note that you can't use braces in the arguments of `\textcolor`.

You can also use a delimited `\color` of the form `\bgroup\color{<color>}<text>\egroup`, a space after that `\egroup` will be considered a space in the output, you don't have to leave a space after the `\egroup` (so `\bgroup\color{red}RedText\egroupOtherText` is valid syntax). You can't nest delimited `\colors`.

Also you can use an undelimited `\color`. It affects anything until the end of the current line (or, if used inside of the `\text` of an delimited `\color`, anything until the end of that delimited `\color`'s `\text`). The syntax would be `\color{color}`.

The package doesn't load anything providing those colouring commands for you and it doesn't provide any coloured animals. The parsing is done using regular expressions provided by L<sup>A</sup>T<sub>E</sub>X3. It is therefore slower than the normal `\AddAnimal`.

### 1.2.2 Options

The following options are available independent on the used code variant (the value of the `version` key). They might be used as package options – unless otherwise specified – or used in the macros `\DucksayOptions`, `\ducksay` and `\duckthink` – again unless otherwise specified. Some options might be accessible in both code variants but do slightly different things. If that’s the case they will be explained in [subsubsection 1.3.3](#) and [subsubsection 1.4.3](#) for `version 1` and `2`, respectively.

$$\text{version}=\langle number \rangle$$

With this you can choose the code variant to be used. Currently 1 and 2 are available. This can be set only during package load time. For a dedicated description of each version look into [subsection 1.3](#) and [subsection 1.4](#). The package author would choose **version=2**, the other version is mostly for legacy reasons. The default is 2.

`\animal` One of the animals listed in [subsection 1.6](#) or any of the ones added with `\AddAnimal`. Not useable as package option. Also don't use it in `\DucksayOptions`, it'll break the default animal selection.

( 3 )

Options.  
For every occasion

```

\      _//|  .-----.
\  _/oo  }          }-@
('')_  }          |
'---'| { }---{ }
      //  /  /  /

```

`animal=<animal>`

Locally sets the default animal. Note that `\ducksay` and `\duckthink` do digest their options inside of a group, so it just results in a longer alternative to the use of `<animal>` if used in their options.

`ligatures=<token list>`

each token you don't want to form ligatures during `\AddAnimal` should be contained in this list. All of them get enclosed by grouping `{` and `}` so that they can't form ligatures. Giving no argument (or an empty one) might enhance compilation speed by disabling this replacement. The formation of ligatures was only observed in combination with `\usepackage[T1]{fontenc}` by the author of this package. Therefore giving the option `ligatures` without an argument might enhance the compilation speed for you without any drawbacks. Initially this is set to `'<>,-'`.

**Note:** In earlier releases this option's expected argument was a regular expression. This means that this option is not fully downward compatible with older versions. The speed gain however seems worth it (and I hope the affected documents are few).

`add-think=<bool>`

by default the animals for `\duckthink` are not created during package load time, but only when they are really used – but then they are created globally so it just has to be done once. This is done because they rely on a rather slow regular expression. If you set this key to `true` each `\AddAnimal` will also create the corresponding `\duckthink` variant immediately.

( 4 )  
 \ .-'\ \  
 " \  
 \_/ ( '-----  
 ,-----)-----)-----)

## 1.3 Version 1

### 1.3.1 Introduction

This version is included for legacy support (old documents should behave the same without any change to them – except the usage of `version=1` as an option). For the bleeding edge version of `ducksay` skip this subsection and read [subsection 1.4](#).

### 1.3.2 Macros

The following is the description of macros which differ in behaviour from those of version 2.

`\ducksay[⟨options⟩]{⟨message⟩}`

options might include any of the options described in [subsection 1.2.2](#) and [subsection 1.3.3](#) if not otherwise specified. Prints an `⟨animal⟩` saying `⟨message⟩`. `⟨message⟩` is not read in verbatim. Multi-line `⟨message⟩`s are possible using `\\`. `\\` should not be contained in a macro definition but at toplevel. Else use the option `ht`.

`\duckthink[⟨options⟩]{⟨message⟩}`

options might include any of the options described in [subsection 1.2.2](#) and [subsection 1.3.3](#) if not otherwise specified. Prints an `⟨animal⟩` thinking `⟨message⟩`. `⟨message⟩` is not read in verbatim. It is implemented using regular expressions replacing a `\` which is only preceded by `\s*` in the first three lines with `0` and `o`. It is therefore slower than `\ducksay`. Multi-line `⟨message⟩`s are possible using `\\`. `\\` should not be contained in a macro definition but at toplevel. Else use the option `ht`.

### 1.3.3 Options

The following options are available to `\ducksay`, `\duckthink`, and `\DucksayOptions` and if not otherwise specified also as package options:

`bubble=⟨code⟩`

use `⟨code⟩` in a group right before the bubble (for font switches). Might be used as a package option but not all control sequences work out of the box there.

`body=⟨code⟩` use `⟨code⟩` in a group right before the body (meaning the `⟨animal⟩`). Might be used as a package option but not all control sequences work out of the box there. E.g. to right-align the `⟨animal⟩` to the bubble, use `body=\hfill`.

`align=⟨valign⟩`

use `⟨valign⟩` as the vertical alignment specifier given to the `tabular` which is around the contents of `\ducksay` and `\duckthink`.

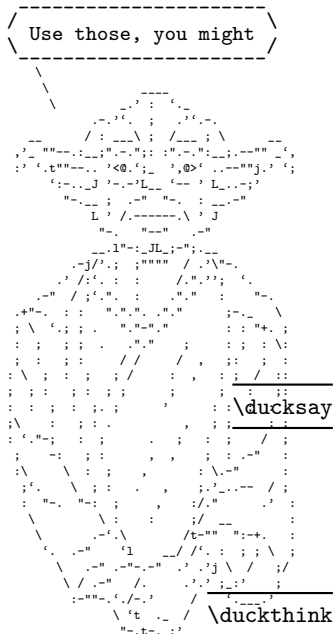
`msg-align=⟨halign⟩`

use `⟨halign⟩` for alignment of the rows of multi-line `⟨message⟩`s. It should match a `tabular` column specifier. Default is `l`. It only affects the contents of the speech bubble not the bubble.

`rel-align=⟨column⟩`

use `⟨column⟩` for alignment of the bubble and the body. It should match a `tabular` column specifier. Default is `l`.

```
( 5 )
\ .-'\V\
" \
- - - - - ( . )
- - - - - ) H H H ) - - - - - H H H H H )
```



**wd**= $\langle count \rangle$  in order to detect the width the  $\langle message \rangle$  is expanded. This might not work out for some commands (e.g. `\url` from `hyperref`). If you specify the width using **wd** the  $\langle message \rangle$  is not expanded and therefore the command *might* work out.  $\langle count \rangle$  should be the character count.

**ht**= $\langle count \rangle$  you might explicitly set the height (the row count) of the  $\langle message \rangle$ . This only has an effect if you also specify **wd**.

```

-----
\ Ohh, no!
-----
\ (.)_(.)
\ (-----)\
/ \ / \ / \
--\ ( ( ( ) ) /--
) \ / \ / \ / \
) _ / / \ / \ \ \

```

### 1.3.4 Defects

- no automatic line wrapping

```

-----
( 6 )
-----
\ .-' \ / \
\ " \ / \
-----
) _ / ( '-----
)-----)-----)-----)

```

## 1.4 Version 2

### 1.4.1 Introduction

Version 2 is the current version of `ducksay`. It features automatic line wrapping (if you specify a fixed width) and in general more options (with some nasty argument parsing).

If you're already used to version 1 you should note one important thing: You should only specify the `version`, the `ligatures` and `add-think` during package load time as arguments to `\usepackage`. The other keys might not work or do unintended things and only don't throw errors or warnings because of the legacy support of version 1.

### 1.4.2 Macros

The following is the description of macros which differ in behaviour from those of version 1.

---

---

`\ducksay`    `\ducksay[⟨options⟩]{⟨message⟩}`

options might include any of the options described in [subsubsection 1.2.2](#) and [subsubsection 1.4.3](#) if not otherwise specified. Prints an `⟨animal⟩` saying `⟨message⟩`.

The `⟨message⟩` can be read in in four different ways. For an explanation of the `⟨message⟩` reading see the description of the `arg` key in [subsubsection 1.4.3](#).

The height and width of the message is determined by measuring its dimensions and the bubble will be set accordingly. The box surrounding the message will be placed both horizontally and vertically centred inside of the bubble. The output utilizes L<sup>A</sup>T<sub>E</sub>X3's coffin mechanism described in [interface3.pdf](#) and the documentation of `xcoffins`.

---

---

`\duckthink`    `\duckthink[⟨options⟩]{⟨message⟩}`

The only difference to `\ducksay` is that in `\duckthink` the `⟨animal⟩`s think the `⟨message⟩` and don't say it.

It is implemented using regular expressions replacing a `\` which is only preceded by `\s*` (any number of space tokens) in the first three lines with `0` and `o`. It's first use per `⟨animal⟩` might therefore be slower than `\ducksay` depending on the `add-think` key (see its description in [subsubsection 1.2.2](#)).

### 1.4.3 Options

In version 2 the following options are available. Keep in mind that you shouldn't use them during package load time but in the arguments of `\ducksay`, `\duckthink` or `\DucksayOptions`.

`arg=⟨choice⟩`

specifies how the `⟨message⟩` argument of `\ducksay` and `\duckthink` should be read in. Available options are `box`, `tab` and `tab*`:

**box** the argument is read in either as a `\hbox` or a `\vbox` (the latter if a fixed width is specified with either `wd` or `wd*`). Note that in this mode any arguments relying on category code changes like e.g. `\verb` will work (provided that you don't use `\ducksay` or `\duckthink` inside of an argument of another macro of course).

**tab** the argument is read in as the contents of a `tabular`. Note that in this mode any arguments relying on category code changes like e.g. `\verb` will *not* work. This mode comes closest to the behaviour of version 1 of `ducksay`.

```
( 7 )
\ .-'\V\
" \
_--/ ( : )
|-----|-----|-----|
```



**tab\***

the argument is read in as the contents of a **tabular**. However it is read in verbatim and uses `\scantokens` to rescan the argument. Note that in this mode any arguments relying on category code changes like e.g. `\verb` will work. You can't use `\ducksay` or `\duckthink` as an argument to another macro in this mode however.

**b** shortcut for `out-v=b`.

`body=<font>` add `<font>` to the font definitions in use to typeset the `<animal>`'s body.

`body*=<font>`  
clear any definitions previously made (including the package default) and set the font definitions in use to typeset the `<animal>`'s body to `<font>`. The package default is `\verbatim@font`. In addition `\frenchspacing` will always be used prior to the defined `<font>`.

`body-align=<choice>`  
sets the relative alignment of the `<animal>` to the `<message>`. Possible choices are `l`, `c` and `r`. For `l` the `<animal>` is flushed to the left of the `<message>`, for `c` it is centred and for `r` it is flushed right. More fine grained control over the alignment can be obtained with the keys `msg-to-body`, `body-to-msg`, `body-x` and `body-y`. Package default is `l`.

`body-mirrored=<bool>`  
if set true the `<animal>` will be mirrored along its vertical centre axis. Package default is `false`. If you set it `true` you'll most likely need to manually adjust the alignment of the body with one or more of the keys `body-align`, `body-to-msg`, `msg-to-body`, `body-x` and `body-y`.

`body-to-msg=<pole>`  
defines the horizontal coffin `<pole>` to be used for the placement of the `<animal>` beneath the `<message>`. See [interface3.pdf](#) and the documentation of `xcoffins` for information about coffin poles..

`body-x=<dimen>`  
defines a horizontal offset of `<dimen>` length of the `<animal>` from its placement beneath the `<message>`.

`body-y=<dimen>`  
defines a vertical offset of `<dimen>` length of the `<animal>` from its placement beneath the `<message>`.

`bubble=<font>`  
add `<font>` to the font definitions in use to typeset the bubble. This does not affect the `<message>` only the bubble put around it.

`bubble*=<font>`  
clear any definitions previously made (including the package default) and set the font definitions in use to typeset the bubble to `<font>`. This does not affect the `<message>` only the bubble put around it. The package default is `\verbatim@font`.

`bubble-bot-kern=<dimen>`  
specifies a vertical offset of the placement of the lower border of the bubble from the bottom of the left and right borders.

`bubble-delim-left-1`= $\langle token list \rangle$   
the left delimiter used if only one line of delimiters is needed. Package default is `(`.

`bubble-delim-left-2`= $\langle token list \rangle$   
the upper most left delimiter used if more than one line of delimiters is needed. Package default is `/`.

`bubble-delim-left-3`= $\langle token list \rangle$   
the left delimiters used to fill the gap if more than two lines of delimiters are needed. Package default is `|`.

`bubble-delim-left-4`= $\langle token list \rangle$   
the lower most left delimiter used if more than one line of delimiters is needed. Package default is `\`.

`bubble-delim-right-1`= $\langle token list \rangle$   
the right delimiter used if only one line of delimiters is needed. Package default is `)`.

`bubble-delim-right-2`= $\langle token list \rangle$   
the upper most right delimiter used if more than one line of delimiters is needed. Package default is `\`.

`bubble-delim-right-3`= $\langle token list \rangle$   
the right delimiters used to fill the gap if more than two lines of delimiters are needed. Package default is `|`.

`bubble-delim-right-4`= $\langle token list \rangle$   
the lower most right delimiter used if more than one line of delimiters is needed. Package default is `/`.

`bubble-delim-top`= $\langle token list \rangle$   
the delimiter used to create the top and bottom border of the bubble. The package default is `{-}` (the braces are important to suppress ligatures here).

`bubble-side-kern`= $\langle dimen \rangle$   
specifies the kerning used to move the sideways delimiters added to fill the gap for more than two lines of bubble height. (the left one is moved to the left, the right one to the right)

`bubble-top-kern`= $\langle dimen \rangle$   
specifies a vertical offset of the placement of the upper border of the bubble from the top of the left and right borders.

`c`  
shortcut for `out-v=vc`.

`col`= $\langle column \rangle$   
specifies the used column specifier used for the  $\langle message \rangle$  enclosing `tabular` for `arg=tab` and `arg=tab*`. Has precedence over `msg-align`. You can also use more than one column this way: `\ducksay[arg=tab,col=cc]{ You & can \\ do & it }` would be valid syntax.

`hpad`= $\langle count \rangle$   
Add  $\langle count \rangle$  times more `bubble-delim-top` instances than necessary to the upper and lower border of the bubble. Package default is 2.

( 9 )

`ht=count` specifies a minimum height (in lines) of the *message*. The lines' count is that of the needed lines of the horizontal bubble delimiters. If the count of the actually needed lines is smaller than the specified *count*, *count* lines will be used. Else the required lines will be used.

`ignore-body=<bool>`  
If set `true` the `<animal>`'s body will be added to the output but it will not contribute to the bounding box (so will not take up any space).

`msg=⟨font⟩` add `⟨font⟩` to the font definitions in use to typeset the `⟨message⟩`.

`msg*=font` clear any definitions previously made (including the package default) and set the font definitions in use to typeset the `message` to *font*. The package default is `\verbatim@font`.

MSG= $\langle font \rangle$  same as msg= $\langle font \rangle$ , bubble= $\langle font \rangle$ .

MSG\*=*font* same as msg\*=*font*, bubble\*=*font*.

`msg-align=<choice>`  
specifies the alignment of the *<message>*. Possible values are l for flushed left, c for centred, r for flushed right and j for justified. If `arg=tab` or `arg=tab*` the j choice is only available for fixed width contents. Package default is l.

`msg-align-c=<token list>`  
 set the `<token list>` which is responsible to typeset the message centred if the option `msg-align=c` is used. It is used independent of the `arg` key. For `arg=tab` and `arg=tab*` the macro `\arraybackslash` provided by `array` is used afterwards. The package default is `\centering`. It might be useful if you want to use `ragged2e`'s `\Centering` for example.

`msg-align-j=<token list>`  
 set the `<token list>` which is responsible to typeset the message justified if the option `msg-align=j` is used. It is used independent of the `arg` key. For `arg=tab` and `arg=tab*` the macro `\arraybackslash` provided by `array` is used afterwards. The package default is empty as justification is the default behaviour of contents of a `p` column and of a `\vbox`. It might be useful if you want to use `ragged2e`'s `\justifying` for example.

`msg-align-l= $\langle token list \rangle$`   
 set the  $\langle token list \rangle$  which is responsible to typeset the message flushed left if the option `msg-align=l` is used. It is used independent of the `arg` key. For `arg=tab` and `arg=tab*` the macro `\arraybackslash` provided by `array` is used afterwards. The package default is `\raggedright`. It might be useful if you want to use `ragged2e`'s `\RaggedRight` for example.

`msg-align-r=<token list>`  
 set the `<token list>` which is responsible to typeset the message flushed right if the option `msg-align=r` is used. It is used independent of the `arg` key. For `arg=tab` and `arg=tab*` the macro `\arraybackslash` provided by `array` is used afterwards. The package default is `\raggedleft`. It might be useful if you want to use `ragged2e`'s `\RaggedLeft` for example.

`msg-to-bubble=<pole>`  
 defines the horizontal coffin `<pole>` to be used as the reference point for the placement of the `<animal>` beneath the `<message>`. See [interface3.pdf](#) and the documentation of `xcoffins` for information about coffin poles..

( 10 )

**none=*bool*** One could say this is a special animal. If **true** no animal body will be used (resulting in just the speech bubble). Package default is of course **false**.

`out-h=<pole>` defines the horizontal coffin `<pole>` to be used as the anchor point for the print out of the complete result of `\ducksay` and `\duckthink`. See [interface3.pdf](#) and the documentation of `xcoffins` for information about coffin poles..

`out-v=\langle pole \rangle` defines the vertical coffin `\langle pole \rangle` to be used as the anchor point for the print out of the complete result of `\ducksay` and `\duckthink`. See [interface3.pdf](#) and the documentation of `xcoffins` for information about coffin poles..

`out-x=dimen`  
specifies an additional horizontal offset of the print out of the complete result of `\ducksay` and `\duckthink`.

`out-y=dimen`  
specifies an additional vertical offset of the print out of the complete result of `\ducksay` and `\duckthink`

`strip-spaces=<bool>`  
if set **true** leading and trailing spaces are stripped from the *<message>* if `arg=box` is used. Initially this is set to **false**.

t                    shortcut for out-v=t.

vpad= $\langle count \rangle$   
 add  $\langle count \rangle$  to the lines used for the bubble, resulting in  $\langle count \rangle$  more lines than necessary to enclose the  $\langle message \rangle$  inside of the bubble.

`wd=<count>` specifies the width of the `<message>` to be fixed to `<count>` times the width of an upper case M in the `<message>`'s font declaration. A value smaller than 0 is considered deactivated, else the width is considered as fixed. For a fixed width the argument of `\ducksay` and `\duckthink` is read in as a `\vbox` for `arg=box` and the column definition uses a `p`-type column for `arg=tab` and `arg=tab*`. If both `wd` is not smaller than 0 and `wd*` is not smaller than 0pt, `wd*` will take precedence.

`wd*= $\langle dimen \rangle$`  specifies the width of the  $\langle message \rangle$  to be fixed to  $\langle dimen \rangle$ . A value smaller than 0pt is considered deactivated, else the width is considered as fixed. For a fixed width the argument of `\ducksay` and `\duckthink` is read in as a `\vbox` for `arg=box` and the column definition uses a p-type column for `arg=tab` and `arg=tab*`. If both `wd` is not smaller than 0 and `wd*` is not smaller than 0pt, `wd*` will take precedence.

$$\left( \begin{array}{c} \text{---} \\ 11 \\ \text{---} \end{array} \right)$$



## 1.6 Available Animals

```

graph TD
    A["( duck )"] --> B["( small-duck )"]
    B --> C["( duck-family )"]
    C --> D["( duck )"]

```

( squirrel )

A line points from the word 'squirrel' to a large, stylized drawing of a squirrel. The drawing is composed of various symbols and characters, including letters, numbers, and punctuation, arranged to form the shape of a squirrel. The squirrel is facing right and has a long tail.

```

graph TD
    A["( C O W )"] --> B["( o o )"]
    A --> C["( _ )"]
    B --> D["| |"]
    B --> E["- w |"]
    C --> F["| -"]
    C --> G["w |"]

```

Diagram illustrating the derivation of the head-in structure from a tree with nodes  $(oo)$  and  $( )$  and a  $w$  node.

```
( pig )
 \ _//| .-----. }-@
  \ _/oo }          |
  (',) _ }          |
   '---| { }---{ }
        // // //
```

[illegible]

```
( snowman )
 \  _[_]_
   (  )
  >-( : )-<
   ( _:_ )
```

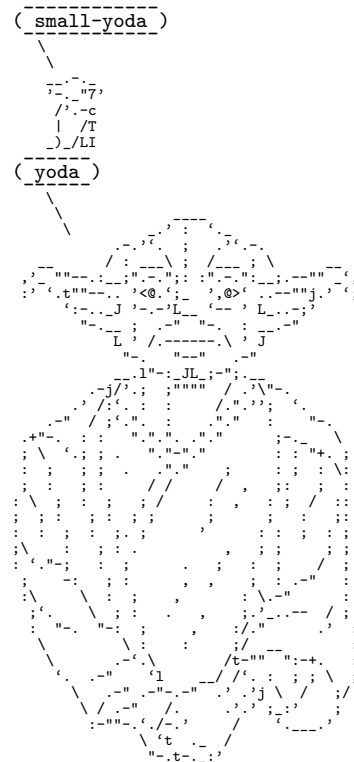
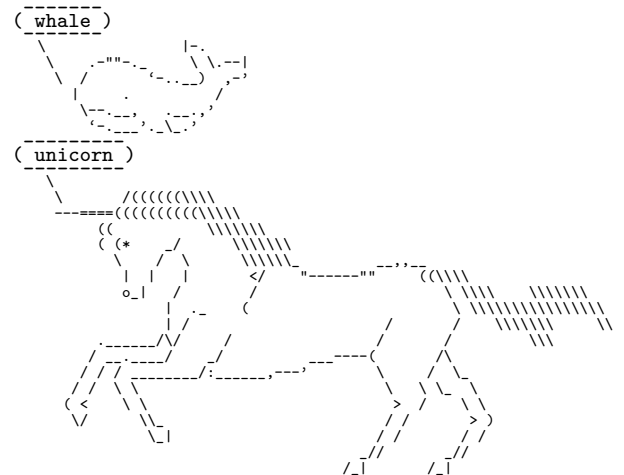
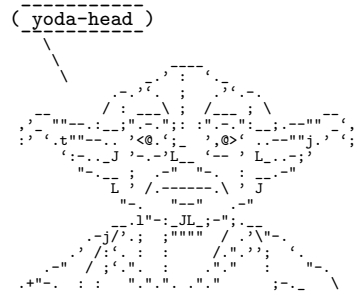
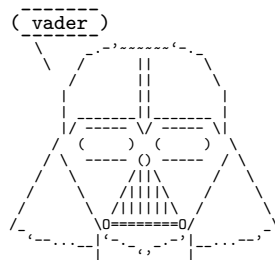
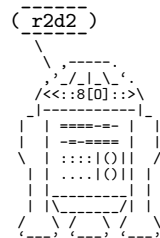
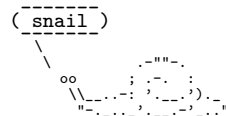
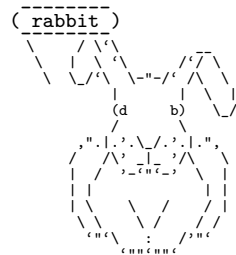
( bunny )  
 \     /  
 ^   ^  
 (   )  
 . ( o ) .

```

( sodomized )
  \
  (oo)\-----/ \ \
  (---)\-----) /
        ||-----w ((
        ||           ||>>

```

```
( hedgehog )
\      .\|//|\\|
\     /|\\|//|//|/
 /   /|\\|//|//|/
  o  /|//|\\|\\|'
```

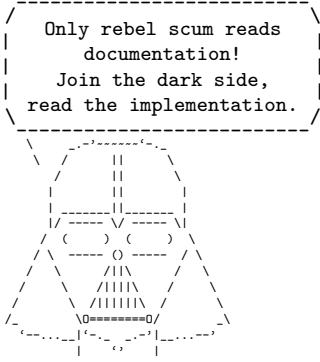


WTFPL would be a better license.

## 1.7 Miscellaneous

This work may be distributed and/or modified under the conditions of the L<sup>A</sup>T<sub>E</sub>X Project Public License (LPPL), either version 1.3c of this license or (at your option) any later version. The latest version of this license is in the file: <http://www.latex-project.org/lppl.txt>

The package is hosted on [https://github.com/Skillmon/ltx\\_ducksay](https://github.com/Skillmon/ltx_ducksay), you might report bugs there.



## 2 Implementation

1 `<*pkg>`

### 2.1 Shared between versions

#### 2.1.1 Variables

##### 2.1.1.1 Integers

2 `\int_new:N \l_ducksay_msg_width_int`  
 3 `\int_new:N \l_ducksay_msg_height_int`

##### 2.1.1.2 Sequences

4 `\seq_new:N \l_ducksay_msg_lines_seq`

##### 2.1.1.3 Token lists

5 `\tl_new:N \l_ducksay_say_or_think_tl`  
 6 `\tl_new:N \l_ducksay_align_tl`  
 7 `\tl_new:N \l_ducksay_msg_align_tl`  
 8 `\tl_new:N \l_ducksay_animal_tl`  
 9 `\tl_new:N \l_ducksay_body_tl`  
 10 `\tl_new:N \l_ducksay_bubble_tl`  
 11 `\tl_new:N \l_ducksay_tmpa_tl`

##### 2.1.1.4 Boolean

12 `\bool_new:N \l_ducksay_also_add_think_bool`  
 13 `\bool_new:N \l_ducksay_version_one_bool`  
 14 `\bool_new:N \l_ducksay_version_two_bool`

##### 2.1.1.5 Boxes

15 `\box_new:N \l_ducksay_tmpa_box`

### 2.1.2 Regular Expressions

Regular expressions for `\duckthink`

16 `\regex_const:Nn \c_ducksay_first_regex { \A(.s*)\\ }`  
 17 `\regex_const:Nn \c_ducksay_second_regex { \A(.^[^c{null}]*c{null}s*)\\ }`  
 18 `\regex_const:Nn \c_ducksay_third_regex {`  
 19 `\A(.^[^c{null}]*c{null}[^c{null}]*c{null}s*)\\ }`  
 20 `\regex_const:Nn \c_ducksay_textcolor_regex`  
 21 `{ \c0(?:\\textcolor\{(.*)\}\{(.*)\}) }`  
 22 `\regex_const:Nn \c_ducksay_color_delim_regex`  
 23 `{ \c0(?:\\bgroup\\color\{(.*)\}(.)\\egroup) }`  
 24 `\regex_const:Nn \c_ducksay_color_regex`  
 25 `{ \c0(?:\\color\{(.*)\}) }`

### 2.1.3 Messages

```
26 \msg_new:nnn { ducksay } { load-time-only }
27 { The~'#1'~key~is~to~be~used~only~during~package~load~time. }
```

### 2.1.4 Key-value setup

```
28 \keys_define:nn { ducksay }
29 {
30   ,bubble .tl_set:N      = \l_ducksay_bubble_tl
31   ,body   .tl_set:N      = \l_ducksay_body_tl
32   ,align  .tl_set:N      = \l_ducksay_align_tl
33   ,align  .value_required:n = true
34   ,wd     .int_set:N      = \l_ducksay_msg_width_int
35   ,wd     .initial:n      = -\c_max_int
36   ,wd     .value_required:n = true
37   ,ht     .int_set:N      = \l_ducksay_msg_height_int
38   ,ht     .initial:n      = -\c_max_int
39   ,ht     .value_required:n = true
40   ,animal .code:n        =
41     { \keys_define:nn { ducksay } { default_animal .meta:n = { #1 } } }
42   ,animal .initial:n      = duck
43   ,msg-align .tl_set:N    = \l_ducksay_msg_align_tl
44   ,msg-align .initial:n   = 1
45   ,msg-align .value_required:n = true
46   ,rel-align .tl_set:N    = \l_ducksay_rel_align_tl
47   ,rel-align .initial:n   = 1
48   ,rel-align .value_required:n = true
49   ,ligatures .tl_set:N    = \l_ducksay_ligatures_tl
50   ,ligatures .initial:n   = { '<>','-' }
51   ,add-think .bool_set:N  = \l_ducksay_also_add_think_bool
52   ,version .choice:
53   ,version / 1 .code:n    =
54     {
55       \bool_set_false:N \l_ducksay_version_two_bool
56       \bool_set_true:N  \l_ducksay_version_one_bool
57     }
58   ,version / 2 .code:n    =
59     {
60       \bool_set_false:N \l_ducksay_version_one_bool
61       \bool_set_true:N  \l_ducksay_version_two_bool
62     }
63   ,version .initial:n     = 2
64 }
65 \ProcessKeysOptions { ducksay }
66
67   Undefine the load-time-only keys
68 \keys_define:nn { ducksay }
69 {
70   version .code:n = \msg_error:nnn { ducksay } { load-time-only } { version }
71 }
```

### 2.1.5 Functions

#### 2.1.5.1 Generating Variants of External Functions

```
70 \cs_generate_variant:Nn \tl_if_eq:nnT { VnT }
```

```
(-----)
\ .-'\V\
"\"
----- (-----)
)-----)-----)-----)-----)
```



## 2.1.5.2 Internal

\duksay\_create\_think\_animal:n

```

71 \cs_new_protected:Npn \duksay_create_think_animal:n #1
72 {
73   \group_begin:
74   \tl_set_eq:Nc \l_duksay_tmpa_tl { g_duksay_animal_say_#1_tl }
75   \regex_replace_once:NnN \c_duksay_first_regex { \10 } \l_duksay_tmpa_tl
76   \regex_replace_once:NnN \c_duksay_second_regex { \1o } \l_duksay_tmpa_tl
77   \regex_replace_once:NnN \c_duksay_third_regex { \1o } \l_duksay_tmpa_tl
78   \tl_gset_eq:cN { g_duksay_animal_think_#1_tl } \l_duksay_tmpa_tl
79   \group_end:
80 }

```

(End definition for \duksay\_create\_think\_animal:n. This function is documented on page ??.)

\duksay\_replace\_verb\_newline:Nn

```

81 \cs_new_protected:Npx \duksay_replace_verb_newline:Nn #1 #2
82 {
83   \tl_replace_all:Nnn #1 { \char_generate:nn { 13 } { 12 } } { #2 }
84 }

```

(End definition for \duksay\_replace\_verb\_newline:Nn. This function is documented on page ??.)

\duksay\_replace\_verb\_newline\_newline:Nn

```

85 \cs_new_protected:Npx \duksay_replace_verb_newline_newline:Nn #1 #2
86 {
87   \tl_replace_all:Nnn #1
88   { \char_generate:nn { 13 } { 12 } \char_generate:nn { 13 } { 12 } } { #2 }
89 }

```

(End definition for \duksay\_replace\_verb\_newline\_newline:Nn. This function is documented on page ??.)

\duksay\_process\_verb\_newline:nnn

```

90 \cs_new_protected:Npn \duksay_process_verb_newline:nnn #1 #2 #3
91 {
92   \tl_set:Nn \ProcessedArgument { #3 }
93   \duksay_replace_verb_newline_newline:Nn \ProcessedArgument { #2 }
94   \duksay_replace_verb_newline:Nn \ProcessedArgument { #1 }
95 }

```

(End definition for \duksay\_process\_verb\_newline:nnn. This function is documented on page ??.)

\duksay\_add\_animal\_inner:nn

```

96 \cs_new_protected:Npn \duksay_add_animal_inner:nn #1 #2
97 {
98   \tl_set:Nn \l_duksay_tmpa_tl { \ #2 }
99   \tl_map_inline:Nn \l_duksay_ligatures_tl
100   { \tl_replace_all:Nnn \l_duksay_tmpa_tl { ##1 } { { ##1 } } }
101   \duksay_replace_verb_newline:Nn \l_duksay_tmpa_tl { \tabularnewline\null }
102   \tl_gset_eq:cN { g_duksay_animal_say_#1_tl } \l_duksay_tmpa_tl
103   \keys_define:nn { duksay }
104   {
105     #1 .code:n =

```

```

( 16 )
\ .-' \
" \
----- ( . ) -----
) -----) -----) -----) -----)

```



```

143   \bool_if:NT \l_ducksay_also_add_think_bool
144     { \ducksay_create_think_animal:n { #2 } }
145   \IfBooleanT{#1}
146     { \keys_define:nn { ducksay } { default_animal .meta:n = { #2 } } }
147   }

```

(End definition for `\AddColoredAnimal`. This function is documented on page 3.)

## 2.1.6 Load the Correct Version and the Animals

```

148 \bool_if:NT \l_ducksay_version_one_bool
149   { \file_input:n { ducksay.code.v1.tex } }
150 \bool_if:NT \l_ducksay_version_two_bool
151   { \file_input:n { ducksay.code.v2.tex } }

152 \ExplSyntaxOff
153 \input{ducksay.animals.tex}

154 </pkg>

```

## 2.2 Version 1

155 `*code.v1`

### 2.2.1 Functions

### 2.2.1.1 Internal

```
\duksay_longest_line:n Calculate the length of the longest line
```

```

156 \cs_new:Npn \ducksay_longest_line:n #1
157 {
158   \int_incr:N \l_ducksay_msg_height_int
159   \exp_args:NNx \tl_set:Nn \l_ducksay_tmpa_tl { #1 }
160   \regex_replace_all:nnN { \s } { \c { space } } \l_ducksay_tmpa_tl
161   \int_set:Nn \l_ducksay_msg_width_int
162     {
163       \int_max:nn
164         { \l_ducksay_msg_width_int } { \tl_count:N \l_ducksay_tmpa_tl }
165     }
166 }

```

(End definition for \ducksay\_longest\_line:n. This function is documented on page ??.)

`\ducksay_open_bubble:` Draw the opening bracket of the bubble

```

167 \cs_new:Npn \ducksay_open_bubble:
168 {
169   \begin{tabular}{@{}l@{}}
170     \null\
171     \int_compare:nNnTF { \l_ducksay_msg_height_int } = { 1 } { ( }
172     {
173       /
174       \int_step_inline:nnn
175         { 3 } { \l_ducksay_msg_height_int } { \\kern-0.2em| }
176       \\detokenize{ \ }
177     }
178     \\[-1ex]\null
179   \end{tabular}
180   \begin{tabular}{@{}l@{}}
181     _\
182     \int_step_inline:nnn { 2 } { \l_ducksay_msg_height_int } { \\ } \\[-1ex]
183     \mbox { - }
184   \end{tabular}
185 }

```

(End definition for \ducksay\_open\_bubble:. This function is documented on page ??.)

`\ducksay_close_bubble:` Draw the closing bracket of the bubble

```

186 \cs_new:Npn \ducksay_close_bubble:
187 {
188   \begin{tabular}{@{}l@{}}
189     _\\
190     \int_step_inline:nnn { 2 } { \l_ducksay_msg_height_int } { \\ } \\[-1ex]
191     { - }
192   \end{tabular}
193   \begin{tabular}{@{}r@{}}
194     \null\\

```

```

195 \int_compare:nNnTF { \l_ducksay_msg_height_int } = { 1 }
196 { ) }
197 {
198   \detokenize {\ }
199   \int_step_inline:nnn
200     { 3 } { \l_ducksay_msg_height_int } { \\|\kern-0.2em }
201   \\/
202 }
203 \\[-1ex]\null
204 \end{tabular}
205 }

```

(End definition for `\ducksay_close_bubble`:. This function is documented on page ??.)

`\ducksay_print_msg:nn` Print out the message

```

206 \cs_new:Npn \ducksay_print_msg:nn #1 #2
207 {
208   \begin{tabular}{@{} #2 @{}}
209     \int_step_inline:nn { \l_ducksay_msg_width_int } { _ } \\
210     #1\\[-1ex]
211     \int_step_inline:nn { \l_ducksay_msg_width_int } { { - } }
212   \end{tabular}
213 }
214 \cs_generate_variant:Nn \ducksay_print_msg:nn { nV }

```

(End definition for `\ducksay_print_msg:nn`. This function is documented on page ??.)

`\ducksay_print:nn` Print out the whole thing

```

215 \cs_new:Npn \ducksay_print:nn #1 #2
216 {
217   \int_compare:nNnTF { \l_ducksay_msg_width_int } < { 0 }
218   {
219     \int_zero:N \l_ducksay_msg_height_int
220     \seq_set_split:Nnn \l_ducksay_msg_lines_seq { \\ } { #1 }
221     \seq_map_function:NN \l_ducksay_msg_lines_seq \ducksay_longest_line:n
222   }
223   {
224     \int_compare:nNnT { \l_ducksay_msg_height_int } < { 0 }
225     {
226       \regex_count:nnN { \c { \\ } } { #1 } \l_ducksay_msg_height_int
227       \int_incr:N \l_ducksay_msg_height_int
228     }
229   }
230   \group_begin:
231     \frenchspacing
232     \verbatim@font
233     \@noligs
234     \begin{tabular}[\l_ducksay_align_tl]{@{}#2@{}}
235       \l_ducksay_bubble_tl
236       \begin{tabular}{@{}l@{}}
237         \ducksay_open_bubble:
238         \ducksay_print_msg:nV { #1 } \l_ducksay_msg_align_tl
239         \ducksay_close_bubble:
240       \end{tabular}\\
241       \l_ducksay_body_tl

```

```

( 20 )
\ .-' \
" \
----- ( .) -----
)-----)-----)-----)-----)

```

```

242         \begin{tabular}{@{}l@{}}
243             \l_ducksay_animal_tl
244         \end{tabular}
245     \end{tabular}
246 \group_end:
247 }
248 \cs_generate_variant:Nn \ducksay_print:nn { nV }

```

(End definition for `\ducksay_print:nn`. This function is documented on page ??.)

`\ducksay_prepare_say_and_think:n` Reset some variables

```

249 \cs_new:Npn \ducksay_prepare_say_and_think:n #1
250 {
251     \int_set:Nn \l_ducksay_msg_width_int { -\c_max_int }
252     \int_set:Nn \l_ducksay_msg_height_int { -\c_max_int }
253     \keys_set:nn { ducksay } { #1 }
254     \tl_if_empty:NT \l_ducksay_animal_tl
255         { \keys_set:nn { ducksay } { default_animal } }
256 }

```

(End definition for `\ducksay_prepare_say_and_think:n`. This function is documented on page ??.)

### 2.2.1.2 Document level

`\ducksay`

```

257 \NewDocumentCommand \ducksay { 0{} m }
258 {
259     \group_begin:
260         \tl_set:Nn \l_ducksay_say_or_think_tl { say }
261         \ducksay_prepare_say_and_think:n { #1 }
262         \ducksay_print:nV { #2 } \l_ducksay_rel_align_tl
263     \group_end:
264 }

```

(End definition for `\ducksay`. This function is documented on page 7.)

`\duckthink`

```

265 \NewDocumentCommand \duckthink { 0{} m }
266 {
267     \group_begin:
268         \tl_set:Nn \l_ducksay_say_or_think_tl { think }
269         \ducksay_prepare_say_and_think:n { #1 }
270         \ducksay_print:nV { #2 } \l_ducksay_rel_align_tl
271     \group_end:
272 }

```

(End definition for `\duckthink`. This function is documented on page 7.)

273 `\</code>.v1`

### 2.3 Version 2

274 `*code.v2`

Load the additional dependencies of version 2.

```
275 \RequirePackage{array,grabbbox}
```

### 2.3.1 Messages

```
276 \msg_new:nnn { ducksay } { justify~unavailable }
```

277 {

```
278 Justified-content-is-not-available-for-tabular-mode-without-fixed-
279 width.~'l'~column-is-used-instead.
```

280 }

```
281 \msg_new:nnn { ducksay } { unknown~message~alignment }
```

282 {

```
283 The-specified-message-alignment-'\exp_not:n { #1 }'-is-unknown.~
284 ']'-is-used-as-fallback.
```

285 }

### 2.3.2 Variables

### 2.3.2.1 Token Lists

```
286 \tl_new:N \l_ducksay_msg_align_vbox_tl
```

### 2.3.2.2 Boxes

```
287 \box_new:N \l_ducksay_msg_box
```

### 2.3.2.3 Bools

```
288 \bool_new:N \l_ducksay_eat_arg_box_bool
```

```
289 \bool_new:N \l_ducksay_eat_arg_tab_verb_bool
```

```
290 \bool_new:N \l_ducksay_mirrored_body_bool
```

#### 2.3.2.4 Coffins

```
291 \coffin_new:N \l_ducksay_body_coffin
```

```
292 \coffin_new:N \l_ducksay_bubble_close_coffin
```

```
293 \coffin_new:N \l_ducksay_bubble_open_coffin
```

```
294 \coffin_new:N \l_ducksay_bubble_top_coffin
```

```
295 \coffin_new:N \l_ducksay_msg_coffin
```

### 2.3.2.5 Dimensions

```
296 \dim_new:N \l_ducksay_hpad_dim
```

```
297 \dim_new:N \l_ducksay_bubble_bottom_kern_dim
```

```
298 \dim_new:N \l_ducksay_bubble_top_kern_dim
```

```
299 \dim_new:N \l_ducksay_msg_width_dim
```

### 2.3.3 Options

```

300 \keys_define:nn { ducksay }
301 {
302   ,arg .choice:
303   ,arg / box .code:n = \bool_set_true:N \l_ducksay_eat_arg_box_bool
304   ,arg / tab .code:n =
305   {
306     \bool_set_false:N \l_ducksay_eat_arg_box_bool
307     \bool_set_false:N \l_ducksay_eat_arg_tab_verb_bool
308   }
309   ,arg / tab* .code:n =
310   {
311     \bool_set_false:N \l_ducksay_eat_arg_box_bool
312     \bool_set_true:N \l_ducksay_eat_arg_tab_verb_bool
313   }
314   ,arg .initial:n = tab
315   ,wd* .dim_set:N = \l_ducksay_msg_width_dim
316   ,wd* .initial:n = -\c_max_dim
317   ,wd* .value_required:n = true
318   ,none .bool_set:N = \l_ducksay_no_body_bool
319   ,body-mirrored .bool_set:N = \l_ducksay_mirrored_body_bool
320   ,ignore-body .bool_set:N = \l_ducksay_ignored_body_bool
321   ,body-x .dim_set:N = \l_ducksay_body_x_offset_dim
322   ,body-x .value_required:n = true
323   ,body-y .dim_set:N = \l_ducksay_body_y_offset_dim
324   ,body-y .value_required:n = true
325   ,body-to-msg .tl_set:N = \l_ducksay_body_to_msg_align_body_tl
326   ,msg-to-body .tl_set:N = \l_ducksay_body_to_msg_align_msg_tl
327   ,body-align .choice:
328   ,body-align / l .meta:n = { body-to-msg = l , msg-to-body = l }
329   ,body-align / c .meta:n = { body-to-msg = hc , msg-to-body = hc }
330   ,body-align / r .meta:n = { body-to-msg = r , msg-to-body = r }
331   ,body-align .initial:n = l
332   ,msg-align .choice:
333   ,msg-align / l .code:n = { \tl_set:Nn \l_ducksay_msg_align_tl { l } }
334   ,msg-align / c .code:n = { \tl_set:Nn \l_ducksay_msg_align_tl { c } }
335   ,msg-align / r .code:n = { \tl_set:Nn \l_ducksay_msg_align_tl { r } }
336   ,msg-align / j .code:n = { \tl_set:Nn \l_ducksay_msg_align_tl { j } }
337   ,msg-align-l .tl_set:N = \l_ducksay_msg_align_l_tl
338   ,msg-align-l .initial:n = \raggedright
339   ,msg-align-c .tl_set:N = \l_ducksay_msg_align_c_tl
340   ,msg-align-c .initial:n = \centering
341   ,msg-align-r .tl_set:N = \l_ducksay_msg_align_r_tl
342   ,msg-align-r .initial:n = \raggedleft
343   ,msg-align-j .tl_set:N = \l_ducksay_msg_align_j_tl
344   ,msg-align-j .initial:n = {}
345   ,out-h .tl_set:N = \l_ducksay_output_h_pole_tl
346   ,out-h .initial:n = l
347   ,out-v .tl_set:N = \l_ducksay_output_v_pole_tl
348   ,out-v .initial:n = vc
349   ,out-x .dim_set:N = \l_ducksay_output_x_offset_dim
350   ,out-x .value_required:n = true
351   ,out-y .dim_set:N = \l_ducksay_output_y_offset_dim

```



```

352 ,out-y .value_required:n = true
353 ,t .meta:n = { out-v = t }
354 ,c .meta:n = { out-v = vc }
355 ,b .meta:n = { out-v = b }
356 ,body* .tl_set:N = \l_ducksay_body_fount_tl
357 ,msg* .tl_set:N = \l_ducksay_msg_fount_tl
358 ,bubble* .tl_set:N = \l_ducksay_bubble_fount_tl
359 ,body* .initial:n = \verbatim@font
360 ,msg* .initial:n = \verbatim@font
361 ,bubble* .initial:n = \verbatim@font
362 ,body .code:n = \tl_put_right:Nn \l_ducksay_body_fount_tl { #1 }
363 ,msg .code:n = \tl_put_right:Nn \l_ducksay_msg_fount_tl { #1 }
364 ,bubble .code:n = \tl_put_right:Nn \l_ducksay_bubble_fount_tl { #1 }
365 ,MSG .meta:n = { msg = #1 , bubble = #1 }
366 ,MSG* .meta:n = { msg* = #1 , bubble* = #1 }
367 ,hpad .int_set:N = \l_ducksay_hpad_int
368 ,hpad .initial:n = 2
369 ,hpad .value_required:n = true
370 ,vpad .int_set:N = \l_ducksay_vpad_int
371 ,vpad .value_required:n = true
372 ,col .tl_set:N = \l_ducksay_msg_tabular_column_tl
373 ,bubble-top-kern .tl_set:N = \l_ducksay_bubble_top_kern_tl
374 ,bubble-top-kern .initial:n = { -.5ex }
375 ,bubble-top-kern .value_required:n = true
376 ,bubble-bot-kern .tl_set:N = \l_ducksay_bubble_bottom_kern_tl
377 ,bubble-bot-kern .initial:n = { .2ex }
378 ,bubble-bot-kern .value_required:n = true
379 ,bubble-side-kern .tl_set:N = \l_ducksay_bubble_side_kern_tl
380 ,bubble-side-kern .initial:n = { 0.2em }
381 ,bubble-side-kern .value_required:n = true
382 ,bubble-delim-top .tl_set:N = \l_ducksay_bubble_delim_top_tl
383 ,bubble-delim-left-1 .tl_set:N = \l_ducksay_bubble_delim_left_a_tl
384 ,bubble-delim-left-2 .tl_set:N = \l_ducksay_bubble_delim_left_b_tl
385 ,bubble-delim-left-3 .tl_set:N = \l_ducksay_bubble_delim_left_c_tl
386 ,bubble-delim-left-4 .tl_set:N = \l_ducksay_bubble_delim_left_d_tl
387 ,bubble-delim-right-1 .tl_set:N = \l_ducksay_bubble_delim_right_a_tl
388 ,bubble-delim-right-2 .tl_set:N = \l_ducksay_bubble_delim_right_b_tl
389 ,bubble-delim-right-3 .tl_set:N = \l_ducksay_bubble_delim_right_c_tl
390 ,bubble-delim-right-4 .tl_set:N = \l_ducksay_bubble_delim_right_d_tl
391 ,bubble-delim-top .initial:n = { { - } }
392 ,bubble-delim-left-1 .initial:n = (
393 ,bubble-delim-left-2 .initial:n = /
394 ,bubble-delim-left-3 .initial:n = |
395 ,bubble-delim-left-4 .initial:n = \c_backslash_str
396 ,bubble-delim-right-1 .initial:n = )
397 ,bubble-delim-right-2 .initial:n = \c_backslash_str
398 ,bubble-delim-right-3 .initial:n = |
399 ,bubble-delim-right-4 .initial:n = /
400 ,strip-spaces .bool_set:N = \l_ducksay_msg_strip_spaces_bool
401 }

```

### 2.3.4 Functions

#### 2.3.4.1 Internal

```

( 24 )
\ .-'\ \
" \
----- ( .) -----
) -----) -----) -----) -----)

```

evaluate\_message\_alignment\_fixed\_width\_common:

```

402 \cs_new:Npn \ducksay_evaluate_message_alignment_fixed_width_common:
403 {
404   \str_case:Vn \l_ducksay_msg_align_tl
405   {
406     { l } { \exp_not:N \l_ducksay_msg_align_l_tl }
407     { c } { \exp_not:N \l_ducksay_msg_align_c_tl }
408     { r } { \exp_not:N \l_ducksay_msg_align_r_tl }
409     { j } { \exp_not:N \l_ducksay_msg_align_j_tl }
410   }
411 }

```

(End definition for \ducksay\_evaluate\_message\_alignment\_fixed\_width\_common:. This function is documented on page ??.)

evaluate\_message\_alignment\_fixed\_width\_tabular:

```

412 \cs_new:Npn \ducksay_evaluate_message_alignment_fixed_width_tabular:
413 {
414   \tl_if_empty:NT \l_ducksay_msg_tabular_column_tl
415   {
416     \tl_set:Nx \l_ducksay_msg_tabular_column_tl
417     {
418       >
419       {
420         \ducksay_evaluate_message_alignment_fixed_width_common:
421         \exp_not:N \arraybackslash
422       }
423       p { \exp_not:N \l_ducksay_msg_width_dim }
424     }
425   }
426 }

```

(End definition for \ducksay\_evaluate\_message\_alignment\_fixed\_width\_tabular:. This function is documented on page ??.)

evaluate\_message\_alignment\_fixed\_width\_vbox:

```

427 \cs_new:Npn \ducksay_evaluate_message_alignment_fixed_width_vbox:
428 {
429   \tl_set:Nx \l_ducksay_msg_align_vbox_tl
430   { \ducksay_evaluate_message_alignment_fixed_width_common: }
431 }

```

(End definition for \ducksay\_evaluate\_message\_alignment\_fixed\_width\_vbox:. This function is documented on page ??.)

\ducksay\_calculate\_msg\_width\_from\_int:

```

432 \cs_new:Npn \ducksay_calculate_msg_width_from_int:
433 {
434   \hbox_set:Nn \l_ducksay_tmpa_box { \l_ducksay_msg_fount_tl M }
435   \dim_set:Nn \l_ducksay_msg_width_dim
436   { \l_ducksay_msg_width_int \box_wd:N \l_ducksay_tmpa_box }
437 }

```

(End definition for \ducksay\_calculate\_msg\_width\_from\_int:. This function is documented on page ??.)

```

(-----)
\ .-' \
" \
----- ( .'------
)-----)

```

\ducksay\_msg\_tabular\_begin:

```

438 \cs_new:Npn \ducksay_msg_tabular_begin:
439 {
440   \ducksay_msg_tabular_begin_inner:V \l_ducksay_msg_tabular_column_tl
441 }
442 \cs_new:Npn \ducksay_msg_tabular_begin_inner:n #1
443 {
444   \begin { tabular } { @{} #1 @{} }
445 }
446 \cs_generate_variant:Nn \ducksay_msg_tabular_begin_inner:n { V }

```

(End definition for \ducksay\_msg\_tabular\_begin:. This function is documented on page ??.)

\ducksay\_msg\_tabular\_end:

```

447 \cs_new:Npn \ducksay_msg_tabular_end:
448 {
449   \end { tabular }
450 }

```

(End definition for \ducksay\_msg\_tabular\_end:. This function is documented on page ??.)

\ducksay\_digest\_options:n

```

451 \cs_new:Npn \ducksay_digest_options:n #1
452 {
453   \keys_set:nn { ducksay } { #1 }
454   \tl_if_empty:NT \l_ducksay_animal_tl
455   { \keys_set:nn { ducksay } { default_animal } }
456   \bool_if:NTF \l_ducksay_eat_arg_box_bool
457   {
458     \dim_compare:nNnTF { \l_ducksay_msg_width_dim } < { \c_zero_dim }
459     {
460       \int_compare:nNnTF { \l_ducksay_msg_width_int } < { \c_zero_int }
461       {
462         \cs_set_eq:NN
463         \ducksay_eat_argument:w \ducksay_eat_argument_hbox:w
464       }
465       {
466         \cs_set_eq:NN
467         \ducksay_eat_argument:w \ducksay_eat_argument_vbox:w
468         \ducksay_calculate_msg_width_from_int:
469       }
470     }
471     {
472       \cs_set_eq:NN \ducksay_eat_argument:w \ducksay_eat_argument_vbox:w
473     }
474   }
475   {
476     \dim_compare:nNnTF { \l_ducksay_msg_width_dim } < { \c_zero_dim }
477     {
478       \int_compare:nNnTF { \l_ducksay_msg_width_int } < { \c_zero_int }
479       {
480         \tl_if_empty:NT \l_ducksay_msg_tabular_column_tl
481         {
482           \str_case:Vn \l_ducksay_msg_align_tl

```

```
\ducksay_set_bubble_top_kern:
```

(End definition for \ducksay\_set\_bubble\_top\_kern:. This function is documented on page ??.)

```
\ducksay set bubble bottom kern:
```

(End definition for \ducksay\_set\_bubble\_bottom\_kern:. This function is documented on page ??.)

\ducksay\_shipout:

```

526 \cs_new_protected:Npn \ducksay_shipout:
527 {
528   \hbox_set:Nn \l_ducksay_tmpa_box
529   { \l_ducksay_bubble_fount_tl \l_ducksay_bubble_delim_top_tl }
530   \int_set:Nn \l_ducksay_msg_width_int
531   {
532     \fp_eval:n
533     {
534       ceil
535       ( \box_wd:N \l_ducksay_msg_box / \box_wd:N \l_ducksay_tmpa_box )
536     }
537   }
538   \group_begin:
539   \l_ducksay_bubble_fount_tl
540   \exp_args:NNNx
541   \group_end:
542   \int_set:Nn \l_ducksay_msg_height_int
543   {
544     \int_max:nn
545     {
546       \fp_eval:n
547       {
548         ceil
549         (
550           (
551             \box_ht:N \l_ducksay_msg_box
552             + \box_dp:N \l_ducksay_msg_box
553           )
554           / ( \arraystretch * \baselineskip )
555         )
556       }
557       + \l_ducksay_vpad_int
558     }
559     { \l_ducksay_msg_height_int }
560   }
561   \hcoffin_set:Nn \l_ducksay_bubble_open_coffin
562   {
563     \l_ducksay_bubble_fount_tl
564     \begin{tabular}{@{}l@{}}
565       \int_compare:nNnTF { \l_ducksay_msg_height_int } = { \c_one_int }
566       {
567         \l_ducksay_bubble_delim_left_a_tl
568       }
569       {
570         \l_ducksay_bubble_delim_left_b_tl\\
571         \int_step_inline:nnn
572         { 3 } { \l_ducksay_msg_height_int }
573         {
574           \kern-\l_ducksay_bubble_side_kern_tl
575           \l_ducksay_bubble_delim_left_c_tl
576           \\
577         }
578         \l_ducksay_bubble_delim_left_d_tl

```

```

579     }
580     \end{tabular}
581 }
582 \hcoffin_set:Nn \l_ducksay_bubble_close_coffin
583 {
584   \l_ducksay_bubble_fount_tl
585   \begin{tabular}{@{}r@{}}
586     \int_compare:nNnTF { \l_ducksay_msg_height_int } = { \c_one_int }
587     {
588       \l_ducksay_bubble_delim_right_a_tl
589     }
590     {
591       \l_ducksay_bubble_delim_right_b_tl \\\
592       \int_step_inline:nnn
593         { 3 } { \l_ducksay_msg_height_int }
594         {
595           \l_ducksay_bubble_delim_right_c_tl
596           \kern-\l_ducksay_bubble_side_kern_tl
597           \\\
598         }
599       \l_ducksay_bubble_delim_right_d_tl
600     }
601   \end{tabular}
602 }
603 \hcoffin_set:Nn \l_ducksay_bubble_top_coffin
604 {
605   \l_ducksay_bubble_fount_tl
606   \int_step_inline:nn { \l_ducksay_msg_width_int + \l_ducksay_hpad_int }
607     { \l_ducksay_bubble_delim_top_tl }
608 }
609 \hcoffin_set:Nn \l_ducksay_msg_coffin { \box_use:N \l_ducksay_msg_box }
610 \bool_if:NF \l_ducksay_no_body_bool
611 {
612   \hcoffin_set:Nn \l_ducksay_body_coffin
613   {
614     \frenchspacing
615     \l_ducksay_body_fount_tl
616     \begin{tabular}{@{}l@{}}
617       \l_ducksay_animal_tl
618     \end{tabular}
619   }
620   \bool_if:NT \l_ducksay_mirrored_body_bool
621   {
622     \coffin_scale:Nnn \l_ducksay_body_coffin
623       { -\c_one_int } { \c_one_int }
624     \str_case:Vn \l_ducksay_body_to_msg_align_body_tl
625       {
626         { l } { \tl_set:Nn \l_ducksay_body_to_msg_align_body_tl { r } }
627         { r } { \tl_set:Nn \l_ducksay_body_to_msg_align_body_tl { l } }
628       }
629   }
630 }
631 \dim_set:Nn \l_ducksay_hpad_dim
632 {

```

```

633      (
634      \coffin_wd:N \l_ducksay_bubble_top_coffin
635      - \coffin_wd:N \l_ducksay_msg_coffin
636      ) / 2
637    }
638    \coffin_join:NnnNnnnn
639    \l_ducksay_msg_coffin      { l } { vc }
640    \l_ducksay_bubble_open_coffin { r } { vc }
641    { - \l_ducksay_hpad_dim } { \c_zero_dim }
642    \coffin_join:NnnNnnnn
643    \l_ducksay_msg_coffin      { r } { vc }
644    \l_ducksay_bubble_close_coffin { l } { vc }
645    { \l_ducksay_hpad_dim } { \c_zero_dim }
646    \ducksay_set_bubble_top_kern:
647    \ducksay_set_bubble_bottom_kern:
648    \coffin_join:NnnNnnnn
649    \l_ducksay_msg_coffin      { hc } { t }
650    \l_ducksay_bubble_top_coffin { hc } { b }
651    { \c_zero_dim } { \l_ducksay_bubble_top_kern_dim }
652    \coffin_join:NnnNnnnn
653    \l_ducksay_msg_coffin      { hc } { b }
654    \l_ducksay_bubble_top_coffin { hc } { t }
655    { \c_zero_dim } { \l_ducksay_bubble_bottom_kern_dim }
656    \bool_if:NF \l_ducksay_no_body_bool
657    {
658      \bool_if:NTF \l_ducksay_ignored_body_bool
659      { \coffin_attach:NVnNVnnn }
660      { \coffin_join:NVnNVnnn }
661      \l_ducksay_msg_coffin \l_ducksay_body_to_msg_align_msg_tl { b }
662      \l_ducksay_body_coffin \l_ducksay_body_to_msg_align_body_tl { t }
663      { \l_ducksay_body_x_offset_dim } { \l_ducksay_body_y_offset_dim }
664    }
665    \coffin_typeset:NVVnn \l_ducksay_msg_coffin
666    \l_ducksay_output_h_pole_tl \l_ducksay_output_v_pole_tl
667    { \l_ducksay_output_x_offset_dim } { \l_ducksay_output_y_offset_dim }
668    \group_end:
669  }

```

(End definition for \ducksay\_shipout:. This function is documented on page ??.)

**2.3.4.1.1 Message Reading Functions** Version 2 has different ways of reading the message argument of \ducksay and \duckthink. They all should allow almost arbitrary content and the height and width are set based on the dimensions.

\ducksay\_eat\_argument\_tabular:w

```

670 \cs_new:Npn \ducksay_eat_argument_tabular:w
671 {
672   \bool_if:NTF \l_ducksay_eat_arg_tab_verb_bool
673   { \ducksay_eat_argument_tabular_verb:w }
674   { \ducksay_eat_argument_tabular_normal:w }
675 }

```

(End definition for \ducksay\_eat\_argument\_tabular:w. This function is documented on page ??.)

```

( 30 )
\ .-'\
" \
----- ( .) -----
) -----) -----) -----)

```

\duksay\_eat\_argument\_tabular\_inner:w

```

676 \cs_new:Npn \duksay_eat_argument_tabular_inner:w #1
677 {
678   \hbox_set:Nn \l_duksay_msg_box
679   {
680     \l_duksay_msg_fount_t1
681     \duksay_msg_tabular_begin:
682     #1
683     \duksay_msg_tabular_end:
684   }
685   \duksay_shipout:
686 }

```

(End definition for \duksay\_eat\_argument\_tabular\_inner:w. This function is documented on page ??.)

\duksay\_eat\_argument\_tabular\_verb:w

```

687 \NewDocumentCommand \duksay_eat_argument_tabular_verb:w
688 { >{ \duksay_process_verb_newline:nnn { ~ } { ~ \par } } +v }
689 { \duksay_eat_argument_tabular_inner:w { \scantokens { #1 } } }

```

(End definition for \duksay\_eat\_argument\_tabular\_verb:w. This function is documented on page ??.)

\duksay\_eat\_argument\_tabular\_normal:w

```

690 \NewDocumentCommand \duksay_eat_argument_tabular_normal:w { +m }
691 { \duksay_eat_argument_tabular_inner:w { #1 } }

```

(End definition for \duksay\_eat\_argument\_tabular\_normal:w. This function is documented on page ??.)

\duksay\_eat\_argument\_hbox:w

```

692 \cs_new_protected_nopar:Npn \duksay_eat_argument_hbox:w
693 {
694   \bool_if:NTF \l_duksay_msg_strip_spaces_bool
695   { \grabbox }
696   { \grabbox* }
697   \l_duksay_msg_box [ \l_duksay_msg_fount_t1 ] \hbox \duksay_shipout:
698 }

```

(End definition for \duksay\_eat\_argument\_hbox:w. This function is documented on page ??.)

\duksay\_eat\_argument\_vbox:w

```

699 \cs_new_protected_nopar:Npn \duksay_eat_argument_vbox:w
700 {
701   \duksay_evaluate_message_alignment_fixed_width_vbox:
702   \bool_if:NTF \l_duksay_msg_strip_spaces_bool
703   { \grabbox }
704   { \grabbox* }
705   [
706     \hsize \l_duksay_msg_width_dim
707     \linewidth \hsize
708     \l_duksay_msg_fount_t1
709     \l_duksay_msg_align_vbox_t1
710     \@afterindentfalse
711     \@afterheading
712   ]

```



```

713     \l_ducksay_msg_box
714     \vbox \ducksay_shipout:
715 }

```

(End definition for \ducksay\_eat\_argument\_vbox:w. This function is documented on page ??.)

### 2.3.4.1.2 Generating Variants of External Functions

```

716 \cs_generate_variant:Nn \coffin_join:NnnNnnnn { NVnNVnnn }
717 \cs_generate_variant:Nn \coffin_attach:NnnNnnnn { NVnNVnnn }
718 \cs_generate_variant:Nn \coffin_typeset:Nnnnn { NVVnn }
719 \cs_generate_variant:Nn \tl_if_eq:nnT { VnT }
720 \cs_generate_variant:Nn \str_case:nn { Vn }
721 \cs_generate_variant:Nn \regex_replace_all:NnN { Nnc }

```

### 2.3.4.2 Document level

**\ducksay**

```

722 \NewDocumentCommand \ducksay { 0{} }
723 {
724   \group_begin:
725   \tl_set:Nn \l_ducksay_say_or_think_tl { say }
726   \ducksay_digest_options:n { #1 }
727   \ducksay_eat_argument:w
728 }

```

(End definition for \ducksay. This function is documented on page 7.)

**\duckthink**

```

729 \NewDocumentCommand \duckthink { 0{} }
730 {
731   \group_begin:
732   \tl_set:Nn \l_ducksay_say_or_think_tl { think }
733   \ducksay_digest_options:n { #1 }
734   \ducksay_eat_argument:w
735 }

```

(End definition for \duckthink. This function is documented on page 7.)

```

736 </code.v2>

```

## 2.4 Definition of the Animals

```

737 <*animals>
738 %^A some of the below are from http://ascii.co.uk/art/kangaroo
739 \AddAnimal{duck}%>>>
740 { \
741   \
742     >(' ')
743     )/
744     /(
745     / '----/
746     \ ~=- /
747     ~~~~~~}%<<<
748 \AddAnimal{small-duck}%>>>
749 { \
750   \
751     >()_
752     (__)__ _}%<<<
753 \AddAnimal{duck-family}%>>>
754 { \
755   \
756     >(' ')
757     )/
758     /(
759     / '----/ -()_ >()_
760     __\__~=-/_ __ (__)__ (__)__ _}%<<<
761 \AddAnimal{cow}%>>>
762 { \ ^__^
763   \ (oo)\_______
764     (__)\       )\/\
765       ||----w |
766       ||     ||}%<<<
767 \AddAnimal{head-in}%>>>
768 { \
769   \ ^__^
770     (oo)\_______/
771     (__)\       )=(  ___|_ \____
772       ||----w |  \ \ \ \____|
773       ||     ||     ||     ||}%<<<
774 \AddAnimal{sodomized}%>>>
775 { \
776   \      ^
777     (oo)\_____/ \ \
778     (__)\       ) /
779       ||----w ((
780       ||     ||>>}%<<<
781 \AddAnimal{tux}%>>>
782 { \
783   \ .--.
784     |o_o |
785     |\\_/_|
786     //  \ \
787     (|    | )

```

```

789      /\_ _/\'
790      \__)=(___/}%<<<
791 \AddAnimal{pig}%>>>
792 + \_ _//| .-----.
793 \_/_oo } }-@
794 (')_ } |
795 '---| { }--{ }
796 //_/ /_/+}%<<<
797 \AddAnimal{frog}%>>>
798 { \
799 \ (.)_(.)
800 - ( - ) -
801 / \/'-----'\ \
802 --\ ( ( ) ) /--
803 ) \ \_./ \ (
804 )_/ /|\ /|\ \_}%<<<
805 \AddAnimal{snowman}%>>>
806 { \
807 \[_]_
808 (")
809 >-( : )-<
810 (:_:_)}%<<<
811 \AddAnimal{hedgehog}%>>>
812 { \ .\|//|\\|\\|
813 \ |/\|//|//|//|
814 /. '|\|\\|//|//|
815 o__._|//|//|\\|\\|}%<<<
816 \AddAnimal{kangaroo}%>>>
817 { \
818 \_-'
819 <--\--/ \
820 \_ / \_ \
821 \, / \ \
822 // \ \
823 ,/' '_,}%<<<
824 %^A http://chris.com/ascii/index.php?art=animals/rabbits
825 \AddAnimal{rabbit}%>>>
826 { \ / \\'
827 \ | \ \' /\' \
828 \ \/' \ -"/' \ \
829 | | \ \ |
830 (d b) \_/
831 / \
832 ,".|.'.\_/.'.|. ",
833 / \\' _|_ \' \ \
834 | / ' " " ' \ |
835 | | | |
836 | \ \ / / |
837 \ \ \ / / /
838 " " \ : /" "
839 " " " " " }%<<<
840 \AddAnimal{bunny}%>>>
841 { \
842 \ /

```

```

843         /\ /
844         ( )
845         .( o ).}%<<<
846 \AddAnimal{small-rabbit}%>>>
847 { \
848     \ _//
849     (')---.
850     _/_ ( )o}%<<<
851 \AddAnimal{dragon}%>>>
852 { \
853     \      | \_ _/ |      / \ // \
854     \      / 0 0 \_ _ / // | \ \
855     /      / \_ _/ // | \ \
856     @_~_@' / \_ _ // | \ \
857     // ^ _/ \_ _ // | \ \
858     ( // ) | \_ _ // | \ \
859     ( / / ) _ | / ) // | \ \
860     ( // / ) ' / _ _ _/ ( ; - . | \ \
861     (( / / ) ) , - { ' . | . . . . . | \ \
862     (( // / ) ) ' \ / \ \ \ \ \ \ \ \ \
863     (( /// ) ) ' . { } / \ \ \ \ \ \ \
864     (( / ) ) . . . . . \ \ \ \ \ \ \
865     /// . . . . . > \ \ \ \ \ \ \
866     /// - . . . . . } ^ - - - - - ~
867                                     / . ~ } %<<<
868 %^~A http://www.ascii-art.de/ascii/def/dogs.txt
869 \AddAnimal{dog}%>>>
870 { \
871     \ .-' \ \
872     " \ '-----.
873     _ _/ ( . '-----
874     ,-----, """,-----, """,-----, """,-----}%<<<
875 %^~A http://ascii.co.uk/art/squirrel
876 \AddAnimal{squirrel}%>>>
877 { \
878     , ;;;;,
879     .-' , ;;;;,
880     /_ ' " = . ' ;;;;,
881     @ = : _ _ , \ , ;;;;,
882     _ ( \ . = ;;;;,
883     ' " _ ( _ / = " '
884     ' " , ' ' } %<<<
885 \AddAnimal{snail}%>>>
886 { \
887     \ .-"-.
888     oo ; .- . :
889     \ \ _ _ . - : ' . _ . ' ) . _
890     " - . _ . _ ' . _ . - ' _ . " } %<<<
891 %^~A http://www.ascii-art.de/ascii/uvw/unicorn.txt
892 \AddAnimal{unicorn}%>>>
893 { \
894     \ /(((((((\ \ \
895     ---====(((((((((((\ \ \
896     (( \ \ \ \ \ \ \

```



```

951 \      .-'\      /t-" " :+ . :
952 ' . -" 'l --/ /' : ; ; \ ;
953 \      .-" .-"-"-' ' ' j \ / ;/
954 \ / .-" / . ' ' ' ;-' ' ;
955 :-"-.' /-' ' / '---'
956 \ 't . _ /
957 "-.t-._:'}%<<<
958 \AddAnimal{yoda-head}%>>>
959 { \
960 \
961 \      : '
962 .-' ' ; ' .-
963 / : ---\ ; /--- ; \
964 ,'_ "----.:_"-" : "----.:_"-' ,
965 :' 't"---.' <@.' ; ,@>' .---"j.' ' ;
966 ':-..J '--'L_ ' L_..;'
967 "-. _ ; .-" "-. : _.-"
968 L ' /-----.\ ' J
969 "-. " " .-"
970 --.l"-:_JL_;" ; _-
971 .-j/' ; ;"" / .'\"-
972 .' /:' : : /."'' ; '
973 .-" / ;' . : ." " : "-.
974 .+"- . : : ". " . " " ;-._ \}%<<<
975 %^A from https://www.ascii-code.com/ascii-art/movies/star-wars.php
976 \AddAnimal{small-yoda}%>>>
977 { \
978 \
979 --.-.-
980 '-."7'
981 /' .-c
982 | /T
983 _)/LI}%<<<
984 \AddAnimal{r2d2}%>>>
985 { \
986 \ ,-----
987 ,'_/_/_\_'
988 /<<:8[0]:>\
989 _|-----|_
990 | | ==-- | |
991 | | --== | |
992 \ | :::( ) | /
993 | | ...| ( ) | |
994 | | -----| |
995 | | \-----/ | |
996 / \ / \ / \
997 '---' '---' '---'}%<<<
998 \AddAnimal{vader}%>>>
999 { \
1000 \ / \ \ \ \ \
1001 / \ \ \ \ \ \
1002 | | | | | |
1003 |-----|-----|
1004 |/ ----- \ / ----- \

```

1005 / ( ) ( ) \  
1006 / \ ----- () ----- / \  
1007 / \ / \ \ / \ \  
1008 / \ / \ \ / \ \  
1009 / \ / \ \ / \ \  
1010 / - \ 0=====0 / - \  
1011 '---...--| '-.-.-.-'|\_---...--' \  
1012 | ' , |}%<<<  
1013 </animals>

Who's gonna use it anyway?

0

o

>( ' )

) /

/(

/ '-----/

\ ~=- /

~ ^ ~ ^ ~ ^ ~ ^ ~ ^ ~ ^ ~ ^

Hosted at  
[https://github.com/Skillmon/ltx\\_ducksay](https://github.com/Skillmon/ltx_ducksay)  
it is.

--.-.-  
'-.-"7'  
/'.-c  
| /T  
\_)\_/\_LI